Data Science: Principles and Practice
Lecture 4: Ensemble Learning

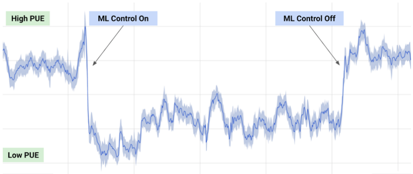
Ekaterina Kochmar

18 November 2019

Wisdom of the crowd

- The collective opinion of a group of individuals rather than that of a single expert
- Classic example: point estimation of a continuous quantity
- 1906 country fair in Plymouth: 800 people participated in a contest to estimate the weight of an ox. Median guess of 1207 pounds accurate within 1% of the true weight of 1198 pounds (Francis Galton)
- Crowd's individual judgments can be modeled as a probability distribution of responses with the median centered near the true value of the quantity to be estimated
- Applications: crowdsourcing, social information sites (Wikipedia, Quora, Stack Overflow), decision-making (trial by jury), sharing economy self-regulating platforms (Uber, Airbnb)

Ensemble-based models in practice



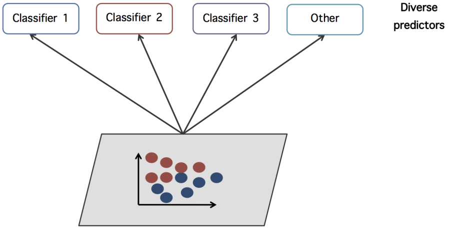
(a) Data centers control by DeepMind uses ensembles of neural networks

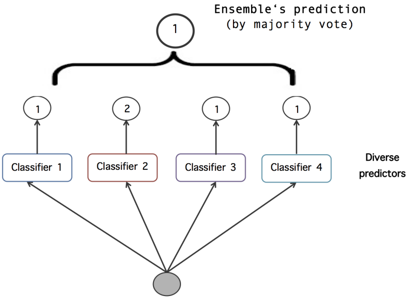
(b) A number of top teams in the competition (`https://netflixprize.com`) used ensembles

Ensemble methods

- Simple voting classifiers using hard and soft voting strategies
- Bagging and pasting ensembles (Random Forests)
- Boosting (AdaBoost, Gradient Boosting)
- Applicable to both classification and regression problems

Voting classifiers



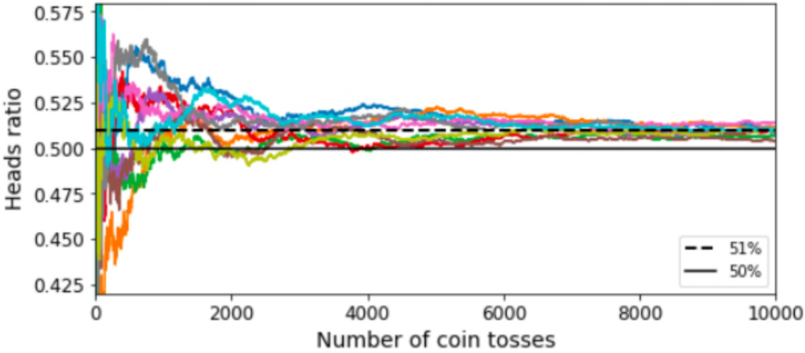


Voting classifier

- Even when individual voters are *weak learners* (hardly above random baseline), the ensemble can still be a *strong learner*
- Condition: individual voters should be sufficiently diverse, i.e. make different (uncorrelated) errors
- Hard to achieve in practice as classifiers usually trained on the same data
- Why does this work?

Coin example

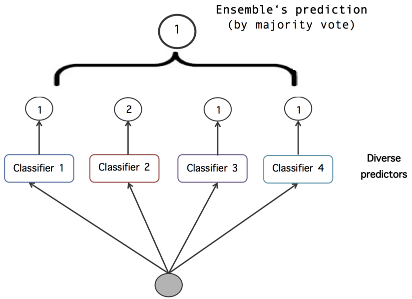
Slightly biased coin: 51% chance of heads



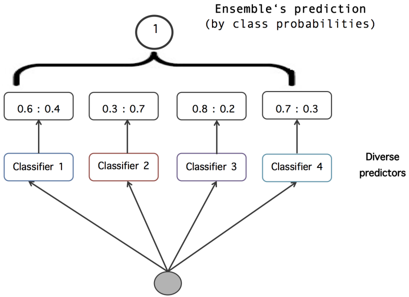
Coin example

- *Law of large numbers*: over the large number of tosses the ratio of heads gets closer to the probability of heads (51%)
- The probability of obtaining the majority of heads after 1000 tosses of this coin approaches 73%; after 10000 tosses – 97%
- If you had 1000 independent classifiers, each of which is only slightly more accurate than random guessing, you can hope to achieve $\sim$ 73%

Hard vs Soft Voting



(c) Hard voting strategy

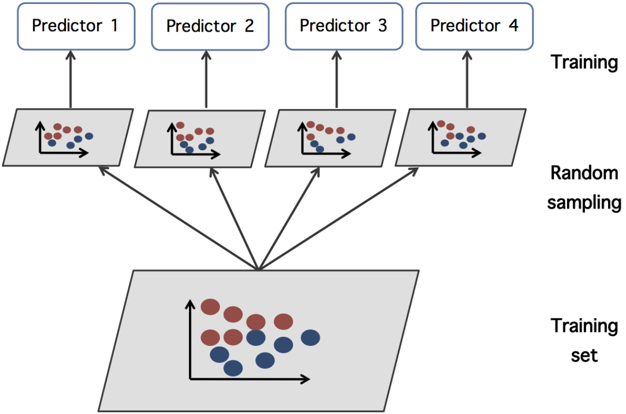


(d) Soft voting strategy

Bagging and Pasting

- One way to ensure that the classifiers decisions are independent is to use very different training algorithms
- Another way – train predictor algorithm on different random subsets of the training data
  - With *bootstrap aggregating* or *bagging* you are sampling *with* replacement
  - With *pasting* you are sampling *without* replacement
- Both strategies allow the predictors to be trained in parallel

Ensembles using bagging



At prediction time, the ensemble makes a prediction, e.g. by taking the *statistical mode* (the most frequent prediction) from the individual predictors

Out-of-Bag Evaluation

- Bagging samples m training instances, where m is the size of the training set
- Only about 63% of the instances are sampled on average for each predictor
- The other 37% are called *out-of-bag* (oob) instances
- Each predictor can be evaluated on the oob instances without any need for a separate validation set or cross-validation

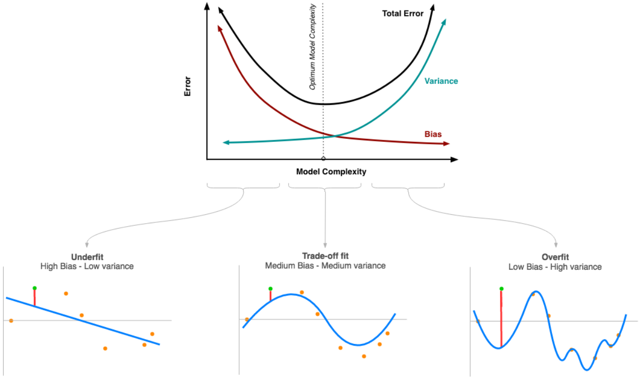
The Bias / Variance Trade-off

Model's generalisation error can be expressed as the sum of three different errors:

1. *Bias* is due to wrong assumptions about the data (e.g. linear instead of quadratic). High-bias model is likely to underfit the training data
2. *Variance* is due to the model's excessive sensibility to small variations in the training data. A model with many degrees of freedom (e.g., high-degree polynomial) is likely to have high variance $\rightarrow$ overfit the training data
3. *Irreducible error* is due to the noisiness in the data itself (solution: clean the data, remove outliers, etc.)

The Bias / Variance Trade-off

Trade-off: Increasing model's complexity will typically increase its variance and reduce bias. Reducing model's complexity increases bias & reduces variance.



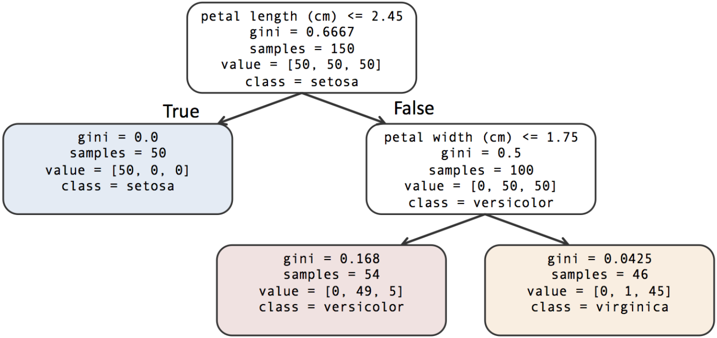
Source: `http://www.ebc.cat/2017/02/12/bias-and-variance/`

What about ensemble models?

- Each individual predictor now has a higher bias than if it were trained on the whole dataset
- Aggregation reduces both bias and variance
- Predictors end up being less correlated, so the ensemble's variance is reduced
- Bagging is generally preferred as it usually results in better models

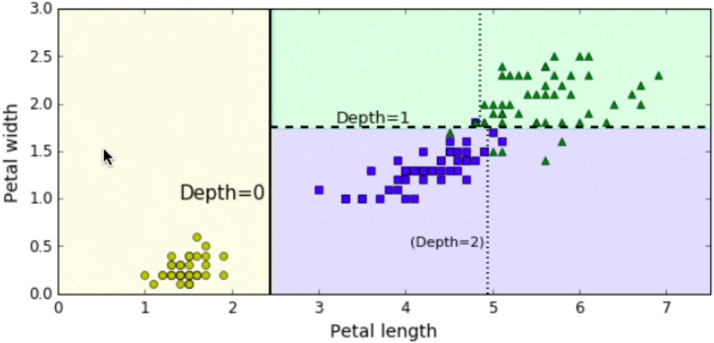
Decision Trees on the *Iris* dataset

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X, y)
```

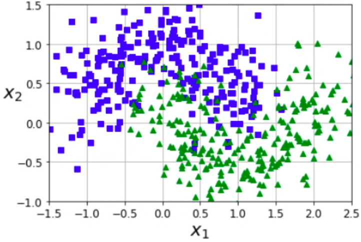


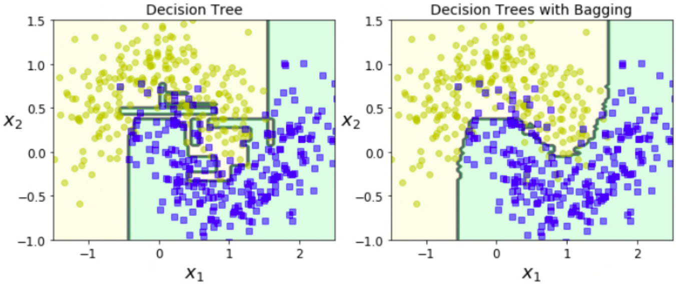
Decision Trees on the *Iris* dataset

Gini (impurity of the node) $= 1 - \sum_{k=1}^{n} p_{i,k}^2$
where $p_{i,k}$ is the ratio of class k instances among the training instances of the i-th node



From a tree to a forest





Random Forests Classifier

- Allows you to control both how the trees are grown (i.e., the typical hyperparameters for Decision Trees) and how the ensemble is built
- Extra randomness: instead of searching for the very best feature to split a node on, it searches for best feature among a random subset of features
- Trading higher bias for lower variance $\rightarrow$ overall, more generalisable
- *Extremely Randomised Trees (Extra-Trees)* use random thresholds for features rather than searching for the best possible thresholds $\rightarrow$ trains much faster

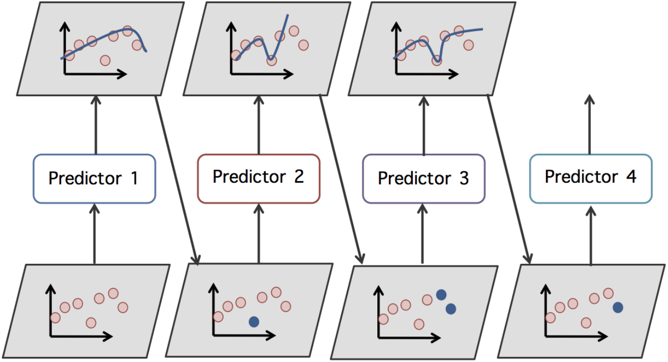
Feature Importance

- Importance of each feature can be measured by looking at how much the nodes that are using a particular feature reduce impurity on the average, i.e. across all trees in the forest
- This can be used for quick understanding of which features matter (feature selection)
- Alternatively, further randomness can be introduced by training on random subsets of the features (supported in `sklearn`):
  - *Random Patches* method when sampling both training instances and features
  - *Random Subspaces* method when keeping all training instances but sampling features

Boosting

- *Boosting* (or *hypothesis boosting*) is an approach that can combine several weaker learners into a stronger learner
- Train predictors sequentially, so that each next classifier tries to correct the errors from its predecessor
- Most popular approaches – `AdaBoost` and `Gradient Boosting`

AdaBoost



- start with the first predictor
- train and estimate performance
- increase relative weight of misclassified training instances
- train new predictor on updated weights and make new predictions
- repeat until stopping criteria are satisfied

AdaBoost

- **Initialisation**: $w^{(i)} = \frac{1}{m}$ for each instance; m – number of instances
- **Error rate**: $r_j = \frac{\sum_{\hat{y}_j^{(i)} \neq y^{(i)}} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$; $\hat{y}_j^{(i)}$ – prediction of j-th classifier on i-th instance
- **Predictor's weight**: $\alpha_j = \eta log \frac{1-r_j}{r_j}$ (higher for more accurate ones); η – learning rate
- **Update**:

$$w^{(i)} = \begin{cases} w^{(i)}, & \text{if } \hat{y}_j^{(i)} = y_j^{(i)} \\ w^{(i)} exp(\alpha_j), & \text{if } \hat{y}_j^{(i)} \neq y_j^{(i)} \end{cases} \tag{1}$$

All instances weights normalised by $\sum_{i=1}^{m} w^{(i)}$

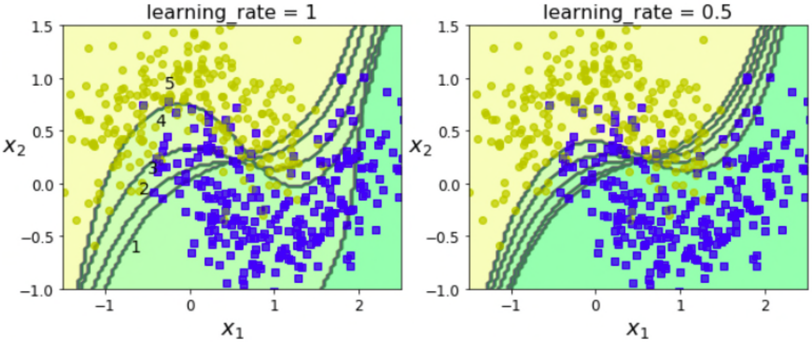
AdaBoost

- **Stopping criteria**: a perfect predictor found, or the pre-defined number of predictors in the ensemble reached
- **Prediction time**:

$$\hat{y}(x) = argmax_k \sum_{j=1;\hat{y}_j(x)=k}^{N} \alpha_j \qquad (2)$$

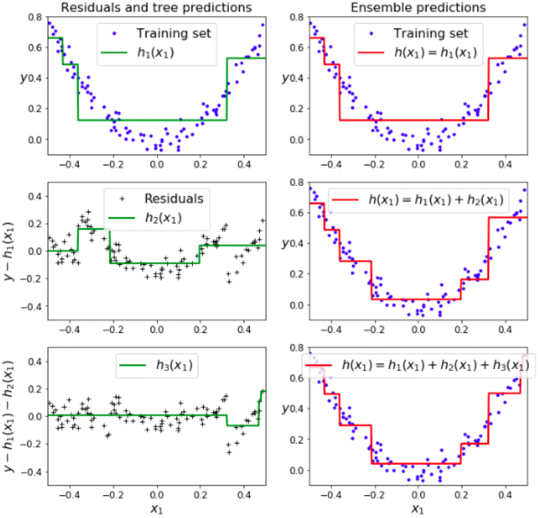
N – number of predictors

AdaBoost with different learning rates



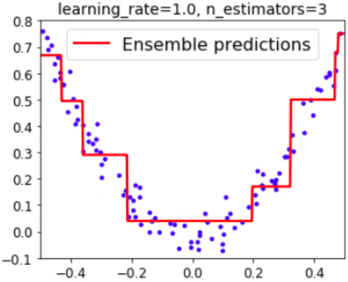
Gradient Boosting

Underlying idea: train predictors on the predecessor's residual errors



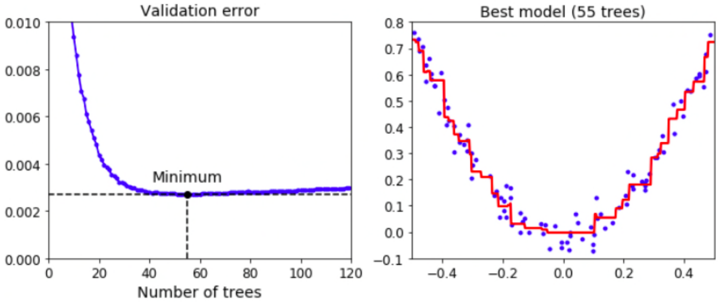
Learning rate

Learning rate scales the contribution of each tree: the lower the rate, the more trees you'll need in the ensemble (but the predictions will usually generalise better)



Early stopping

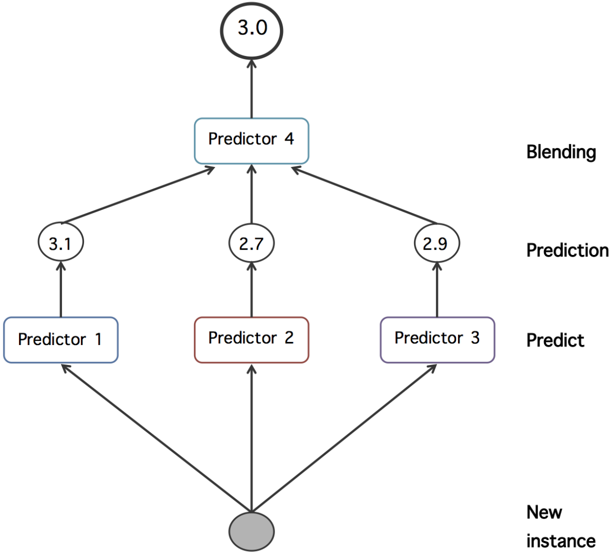
Q: How do we know when to stop?
A: Estimate validation error and stop when it reached a minimum (or does not improve for a number of iterations)



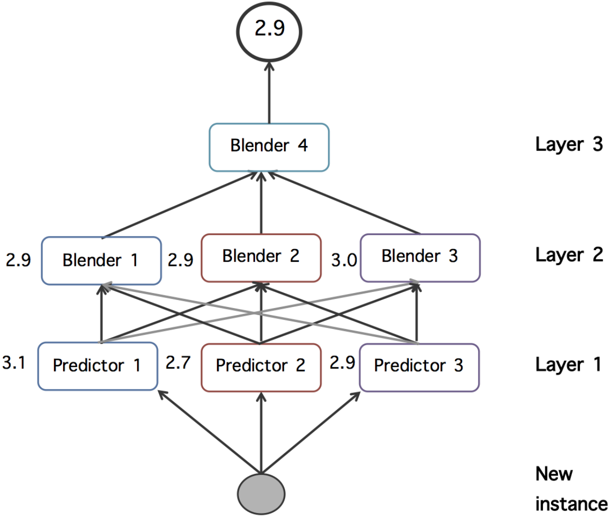
Stacking

- *Stacking* (*stacked generalisation*) – instead of using a trivial function like hard voting to aggregate predictions, why not train a model to learn such aggregation?
- This model is called *blender* or *meta-learner*

Stacking


3.0
Predictor 4
Blending
3.1
2.7
2.9
Prediction
Predictor 1
Predictor 2
Predictor 3
Predict
New
instance

Multi-layer stacking ensemble



Practical 3: Ensemble-based learning

Your tasks

- Run the code in the notebook. During the practical session, be prepared to discuss the methods and answer the questions
- Apply ensemble techniques of your choice to one of the datasets you've worked on during the previous practicals and report your findings
- **Optional**: implement stacking algorithm